Spoken Natural Language Agents for ModSAF Query and Reference in QuickSet*

Kenneth Wauchope
Navy Center for Applied Research in Artificial Intelligence
US Naval Research Laboratory, Code 5512
Washington, DC 20375-5337

ABSTRACT

This report describes a set of speech input, natural language processing and speech output agents that add ModSAF database query and descriptive reference capability to Oregon Graduate Institute's QuickSet multimodal interface to a military simulation system.

INTRODUCTION

QuickSet [Cohen 1997] is a multimodal interface that has been integrated into the Marine Corp's LeatherNet training simulator for instantiating new ModSAF entities and controlling a 3D stealth viewer. Entity creation actions consist typically of a spoken noun phrase accompanied by a mouse/pen gesture on the map display indicating the object's desired position and/or size and orientation, for example the utterance *M1A1 platoon* accompanied by a single mouse click, or *Line of departure* accompanied by a linear gesture. Military units can also have callsigns associated with them (*M1A1 platoon called whiskey four six*). Movement commands to ModSAF are of the form *M1A1 platoon follow this route* accompanied by a linear gesture. Finally, the stealth viewer can be issued commands such as *CommandVU pitch up* and *CommandVU fly me to Objective Bravo*.

It should be noted that these commands all represent imperatives to perform various actions, and so do not include means for querying the current state of the simulation. In addition, the referential capability of the interface is restricted to simple class designations or names (*M1A1 platoon*, *Objective Bravo*) without descriptive referential capabilities such as quantification and relativization ("all platoons north of here").

The Navy Center for Applied Research in Artificial Intelligence has been involved for a number of years in building multimodal interfaces to military systems, most recently the InterLACE [Wauchope 1996] interface to a map-based Air Force combat simulation. The similarity of application domain made it clear that the cartographic database query and extended reference capabilities developed in InterLACE could transition quite naturally to QuickSet. In addition, QuickSet is based on an agent-based architecture (SRI International's Open Agent Architecture [Cohen 1994]) that held the promise of our being able to add new speech and language agents to QuickSet without any recoding or recompiling of the original system.

The project described in this report had three main objectives. The first was to convert the Navy AI Center's NAUTILUS natural language processor [Wauchope 1994, 1997] to an OAA agent capable of taking over from the built-in QuickSet NL agent. The second was to extend the NAUTILUS-based NL interface to include database queries and extended reference capability. Finally, the enhanced NL input requirements required that we develop our own textual sentence input and speech input agents, as well as a speech output agent for making verbal query responses.

---

* AIC Technical Report AIC-00-004. This work was sponsored by the Office of Naval Research.

AGENTIFYING NAUTILUS

In QuickSet, the various I/O processing tasks (speech recognition, NL processing, gesture recognition, and multimodal integration) are handled by individual agent processes that communicate via a central blackboard or facilitator agent using a common Prolog-based Interagent Communication Language (ICL).

Converting an application to an OAA agent requires linking to SRI's Agent Library (`agentlib.c`) and its TCP-based support code, and providing function definitions for three stub functions, most importantly a function `DO_EVENT` which is responsible for formatting and transmitting output messages to the blackboard agent. At startup the application registers with the blackboard via an OAA function call, and then runs an OAA polling loop function to receive and process incoming messages from the blackboard.

NAUTILUS is written in Common LISP, which in most implementations has a foreign function capability allowing the linking in of external C object code. We had previously used our own TCP-based library for communications between NAUTILUS and target applications, so the fundamentals of the Agent Library approach were quite similar. While the process of acquiring a copy of the Agent Library languished in legal difficulties for a number of months, we were finally able to obtain a Public Domain copy of an early version of the source code that had been included in a ModSAF software distribution. Since the Agent Library code was heavily UNIX-oriented, we first used it to agentify a version of NAUTILUS running in Lucid Common LISP on Sun Workstations under SunOS 4.x.

OAA agents register with the blackboard by announcing what types of messages they wish to receive. In the case of the QuickSet NL agent (OGI_NL), the only message it responds to is of the form `parse_speech(_)`, representing a request to parse a spoken text that has being posted on the blackboard by the speech recognition agent. For example the speech input "barbed wire" results in the blackboard relaying the following ICL message to OGI_NL:

```
parse_speech(speech('BARBED WIRE',
                authent(wauchope,pda95,[0,148],pentium,wauchope),
                interval(timeval(901294928,76),timeval(901294932,41)),
                1.0E+000,
                [s,2,1]))
```

This represents a request to parse a speech event consisting of the text `'BARBED WIRE'` received from an authenticated user, uttered during a particular 4-second time interval with probability 100%, and identified as the first decoding of the second input sentence of the session. OGI_NL's output after parsing this utterance is a `speech_list` message:

```
speech_list (speech('BARBED WIRE',
             [fsTYPE:create_line,
                  object:[fsTYPE:barbed_wire_obj,
                       style:barbed_wire,
                       color:red,
                       label:'Barbed Wire'|_6580],
                  location:[fsTYPE:line|_6591]|_6586],
            authent(wauchope,pda95,[0,148],pentium,wauchope),
            interval(timeval(901294928,76),timeval(901294932,41)),
            1.0E+000,
            partial,
            [s,2,1]))
```

The first new argument in the `speech` operand is a typed feature structure (`fsTYPE`) representing the logical form of the parse, in this case a command to create a line object of a particular graphical style and color, labeled `'Barbed Wire'`, and located at some currently unbound linear extent. The second new field is a status (`partial`) indicating that the FS is incomplete and must be unified with a gestural input to provide a binding for the location. The QuickSet multimodal integration agent can then unify the location field of this partial input with the sequence of UTM coordinates output by the gesture recognition agent to produce a complete command for QuickSet to execute, resulting in a new barbed wire entity being drawn at the specified location on the map.

NAUTILUS DATA EXTENSIONS

Our NAUTILUS grammars already contained a sentence fragment production allowing a bare noun phrase like *barbed wire* to be interpreted as having an implicit zeroed predicate (i.e. "create"), so all that was needed was to provide semantic case frames mapping the null predicate to the appropriate feature structure type (`create_line`, `create_area`, `create_point`, `create_unit`, `exinit_create_unit`, `create_feature_object` and `create_feature_line`) required by the multimodal integrator. For *Barbed wire* this results in the NAUTILUS logical form

```
(COMMAND (FORALL X3 (SETOF N1 SYSTEM)
          (EXISTS! X4 (SETOF N2 BARBED_WIRE)
                   (CREATE_LINE :AGENT X3 :THEME X4))))
```

representing a command to the system to create a line using `N2` (an object in NAUTILUS's FOCAL reference resolution component representing a generic barbed wire entity) as model. The Translation Function `CREATE_LINE` is in turn defined as

```
(defun CREATE_LINE (&key AGENT THEME)
   (make-fsTYPE
        :type 'create_line
        :features
        (list :object (make-fsTYPE :type (get theme :class)
                                   :features (FOCAL-features theme))
              :location (make-fsTYPE :type 'line))))
```

where function `make-fsTYPE` constructs the appropriate feature structure string

```
[fsTYPE:create_line,object:[fsTYPE:barbed_wire_obj,style:barbed_wire,
   color:red,label:'Barbed Wire'|_],location:[fsTYPE:line|_]|_]
```

by obtaining the necessary types and features (`barbed_wire_obj`, etc.) from the FOCAL extensional object passed in as `THEME`. This string is then used to create the final `speech_list` message for transmission to the QuickSet blackboard agent by the `DO_EVENT` internal event handler.

Rather than attempt to duplicate the entire OGI_NL object creation capability in this phase of the project, we elected to implement a lexicon, semantics and reference model for just a few instances of each object type (line, area, point, unit, EXINIT unit, feature object and feature line) as a proof of feasibility. It was then possible to start up QuickSet on a networked Windows PC, kill its OGI_NL language agent, start up the NAUTILUS language agent (NRL_NL) on a Sun Workstation, and have it take over language processing seamlessly and transparently.

WINDOWS AGENTLIB PORT

We next succeeded in modifying the UNIX Agent Library source code to compile under Windows, allowing us to agentify our PC version of NAUTILUS running under Allegro Common LISP 5.0. This was particularly important since our Suns were in the process of being upgraded to Solaris 2.7 and the UNIX agent library would not run successfully in GNU Common LISP, which at the time was our only fully Solaris-compatible CL (since then we have acquired a copy of Allegro Common LISP 5.0.1 for Solaris, in which the Agent Library does run successfully).

QUERY AND REFERENCE CAPABILITY

Our next objective was to port a number of the linguistic query and referential capabilities of InterLACE to the QuickSet application.

The first step was to determine what queriable attributes or relationships exist in the ModSAF objects created by QuickSet. Aside from the type specifier (allowing existential queries such as *Are there any barbed wires?* and *How many barbed wires are there?*), most of the intrinsic attribute information in these objects is graphical in nature and serves to determine such things as what line style and color to use in drawing them. The only exceptions we found were that command post objects have three attributes representing the number of jeeps, volunteers and helicopters at the post, and certain cultural feature objects (hospital, mosque, church and school) have an attribute representing the number of residents. In both cases, however, QuickSet does not allow these values to be specified by the user but just fills in its own default values when the object is instantiated, making them features unlikely to be queried by a user.

While these objects have few intrinsic nongraphical attributes, their point locations and extents can be used to derive a number of geometric attributes (length, area) and relationships (distance, direction, intersection) that are potentially informative to the user. Combined with a variety of universal, definite and indefinite quantifiers (*every, the, a*), anaphoric expressions (*it, they, that*) and deictic references (*this, here*), such queries as the following then become possible:

How long is that wire?
Is it shorter than this ditch?
What direction is the M1A1 platoon from the command post?
How far apart are they?
Which platoon is closest to here?
Does every barbed wire cross a berm?

The queriable predicates can also be incorporated into descriptive references using relativization:

What's the longest wire that crosses this ditch?

in effect merging multiple queries into one, i.e., which wires cross this ditch and which of them is longest.

IMPLEMENTATION

Quantifier and anaphor handling is already built into NAUTILUS and the syntax and declarative semantics of the relevant geometric attributes and relationships had already been worked out in InterLACE, so establishing a referential and query interface between NAUTILUS and QuickSet required developing three new pieces of code: (1) noun-dereferencing routines for retrieving sets of database objects of specified types, (2) routines for retrieving the values of geometric attributes of objects and relationships between objects, and (3) a mechanism for handling deixis, or verbal reference to entities denoted graphically in the QuickSet map display.

Noun Dereferencing

QuickSet's ModSAF stub agent conforms to an Agent Layer Language that provides a number of ICL message types for querying the ModSAF database. For example, the query

```
modsaf(list_entities(_))
```

might return the database response

```
[modsaf(list_entities(entity([0,0,4],'100A1',unit_USSR_T72M_Company,
                        utm(51,'S','VL',17790,94370),
                        otherForceID,[0,0,0],[0,0,0],[0,0,0],
                        0,360,false,false,2.5E-001))),
 modsaf(list_entities(entity([0,0,6],'100th',unit_US_armored_Company,
                        latlon(36991039,122062201),
                        distinguishedForceID,[0,0,0],[0,0,0],[0,0,0],
                        0,0,false,false,2.5E-001)))]
```

representing two ModSAF units conforming to the following argument structure:

```
entity(POID, Marking, Type, UTM, Force, Commander, TaskFrame, Overlay,
       Appearance, ...))
```

In this case the `Type` field is sufficient to identify a unit of a particular class, so NAUTILUS can convert the NL reference *T72 companies* to the database query

```
modsaf(list_entities(_,_,unit_USSR_T72M_Company,_,...))
```

which would retrieve only the first of the units above. The returned value then serves as the local extension of the noun phrase for logical form evaluation, as well as an antecedent for possible subsequent anaphoric followups such as *How far is it/that company from the command post?*

In the case of line objects (lines, areas and minefields) and EXINIT point objects (including cultural feature points), however, a single database access such as the above is not sufficient. For example, the query `modsaf(list_lines(_))` might return the database response

```
(line([0,0,4], 8,
     [utm(51,'S','VL',16460,94387),utm(51,'S','VL',16459,94306)...],
     noArrowHead, noArrowHead,'OCRed',true,false,false,
     'LStempplain',2,10,[0,0,0]))
```

where the graphical `Style 'LStempplain'` is insufficient to uniquely identify this object as being a Line of Departure, Phase Line, Swamp, or some other area object. To retrieve the set of all Lines of Departure, then, we must additionally query a separate `TEXT` object created by QuickSet and unify its associated object POID (unique Persistent Object identifier) with the POID of a line object:

```
(modsaf(list_texts(text(_,POID,'LOD')),
        list_lines(line(POID,_,_,_,_,_,_,_,_,_,_,_,_))))
```

Since ModSAF objects are not hierarchically typed, it is not possible to query simply for "companies" -- one must query individually for `unit_USMC_M1A1_Company`, `unit_USSR_T72M_Company`, etc. NAUTILUS handles this problem by dereferencing "companies" as the set of generic FOCAL objects `M1A1_COMPANY`, `T72_COMPANY`, etc. and then querying the ModSAF database individually for each of their types.

Command Posts are point objects with `Name 'Command post'`, so their access is straightforward. Objectives and checkpoints are point objects whose name is a catenation of the abbreviated type plus an obligatory designator (`'Cp blue'`, `'Obj Owl'`), so for example to access all checkpoints requires first accessing all point objects and then filtering out those with a "Cp"-headed name.

Attribute/Relation Evaluation

NAUTILUS normalizes synonymous attribute queries such as *How long are the wires?*, *What length are all the wires?*, and *What is the length of each wire?* into the same logical form

```
(TELL (SETOF X11 (SETOF N9 BARBED_WIRE)
             (SETOF X12 (SETOF N10 LENGTH)
                    (HAVE-LENGTH :CARRIER X11 :POSSESSED X12))))
```

[tell me for each barbed wire what length it has]. The first `SETOF` operator takes each barbed wire object retrieved from the ModSAF database and invokes a `HAVE-LENGTH` Translation Function on it. `HAVE-LENGTH`, in turn, invokes a so-called Interface Function `COMPUTE-LENGTH` which scans the object's `Points` field (a list of UTM coordinates) to compute the object's length in meters. `HAVE-LENGTH` converts this result into an appropriate format ("900 meters", "1.5 kilometers") which it passes back to the `TELL` performative for composing into a final query response, e.g. "Wire number 1 length 900 meters, and wire number 2 length 1.5 kilometers".

Similar Translation Functions exist for direction (*What direction is <x> from <y>?*) and distance (*How far is <x> from <y>?*). For example the Translation Function `HAVE-DIRECTION` is defined as

```
(defun HAVE-DIRECTION (&key carrier from-loc)
   (elt '("west" "southwest" "south" "southeast"
          "east" "northeast" "north" "northwest")
        (compute-direction carrier from-loc)))
```

where the Interface Function `COMPUTE-DIRECTION` returns an integer from 0 to 7 representing one of eight cardinal directions. Direction predicates `SOUTHWEST`, `SOUTH`, etc. (*Is <x> south of <y>?*) are defined in terms of the same Interface Function:

```
(defun SOUTHWEST (&key theme cotheme)
   (eq (compute-direction theme cotheme) 1))

(defun SOUTH (&key theme cotheme)
   (<= 1 (compute-direction theme cotheme) 3))
```

In this case, point object <x> is defined as being south of point object <y> if its orientation to <y> is between southwest and southeast inclusive. We make no claim that this particular procedural semantics is ideal, only that it could be easily redefined in the Translation and/or Interface Function without affecting the rest of the system. The procedural semantics for the orientation of a point to a line object is even more

perfunctory (we take its orientation to the midpoint of the line), and is not defined at all between line and/or area objects at present.

The `COMPUTE-DISTANCE` Interface Function (*How far is <x> from <y>?*) just looks for the smallest distance between a point in <x>'s coordinate list and a point in <y>'s coordinate list, and so does not investigate whether an intermediate point on the line segment between two coordinates might be closer (it does, however, report the distance between two intersecting line objects as zero). The `LINES-CROSS` Interface Function (*Does <x> cross <y>?*) uses a line segment intersection algorithm to determine if the segment between any pair of points in line <x> intersects the segment between any pair of points in line <y>.

Comparatives (*longer*, *shorter*, *closer*, *farther*) are also defined using the `COMPUTE-LENGTH` and `COMPUTE-DISTANCE` functions, for example

```
(defun CLOSER (&key theme loc cotheme)
  (and (not (equal theme loc))
       (< (compute-distance theme loc)(compute-distance cotheme loc))))
```

Finally, NAUTILUS automatically converts superlatives (*longest*, *closest*) into logical forms involving their respective comparatives, e.g. *Which platoon is closest to the wire? =*

```
(TELL (SETOF X14 (SETOF N11 PLATOON)
           (FORALL X16 (REMOVE X14 N11 :TEST #'STRING=)
                  (EXISTS! X17 (SETOF N1 BARBED_WIRE)
                           (CLOSER :THEME X14 :LOC X17 :COTHEME X16)))))
```

(tell the set of platoons which are closer to the wire than all the other platoons).

The procedural semantics for the geometric Interface Functions are implemented as foreign code in C so as to more easily access the various ICL parsing and argument extraction routines available from the Agent Library.

Deictic Reference

Deictic references in the original QuickSet system take two main forms. First, one can draw a route on the screen while saying *Follow this route*. Hence this is not a reference to an already existing graphical object, but is really a multimodal object creation action (as if one were to say "This is barbed wire" or "Barbed wire here" when drawing a line instead of just *Barbed wire*). Similarly, in Multiple Creation mode the user can use deictic followups, as in *Multiple M1 platoons (click) here (click) here (click)* where again it is as if the user were saying "M1 platoon here".

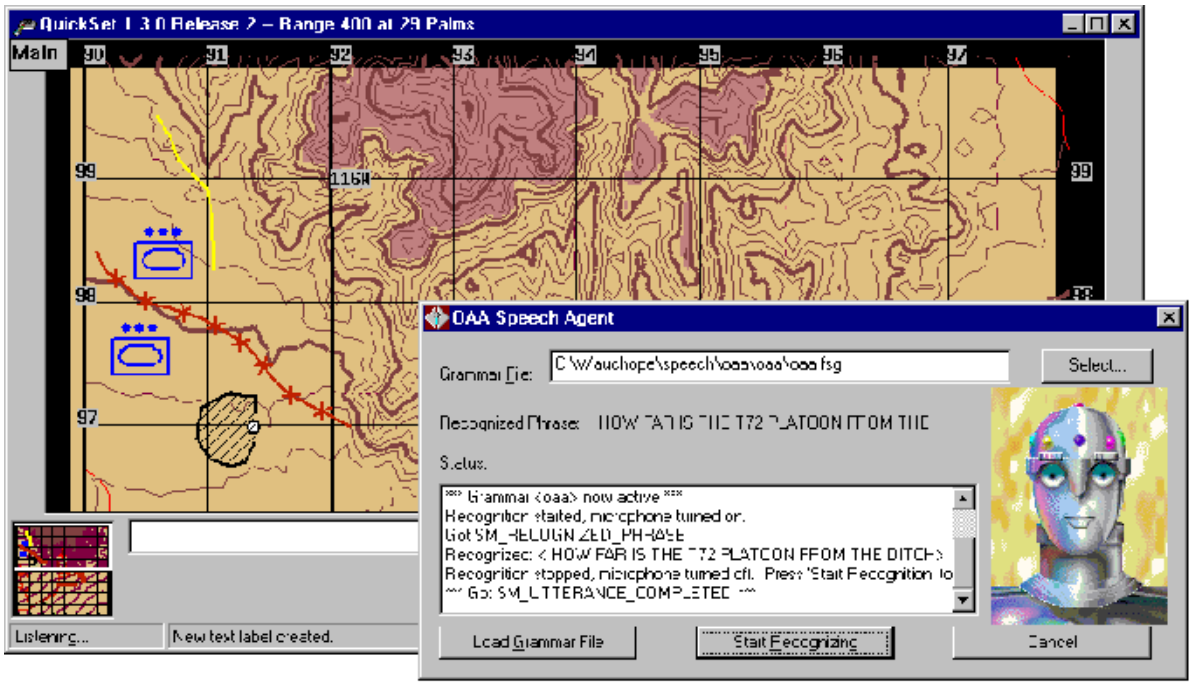
Second, the CommandVU interface allows inputs such as *CommandVU take me here (click)* and *CommandVU take me to this platoon (gesture near object)*, where in the first case the click just signifies a coordinate location but in the second it is interpreted as signifying the platoon nearest to where the user gestured. The latter is closer to the kind of deictic reference used in NRL's Eucalyptus [Wauchope 1994] interface, where a click near a tanker object could accompany a verbal reference such as *Have the F14 refuel here/at this tanker.* In InterLACE, on the other hand, we chose a different multimodal treatment of deixis in which mouse clicks resulted in graphical object selections (as in a conventional GUI), and deictic expressions such as *this* or *these towns* were then interpreted as references to subsets of the selected entities.

Although the QuickSet map does not provide any direct manipulation capability, objects are in fact selected or deselected when the mouse is held over them for a second, and using control keys one can perform multiple object selections as well. So while it probably would have been possible to intercept the blackboard postings from the QuickSet gesture recognizer and implement a gesture-based deixis mechanism in NAUTILUS similar to the Eucalyptus approach, we elected instead to use the QuickSet selection capability and implement an approach similar to that taken in InterLACE. The user then either holds the mouse for a second over an entity while making a query (e.g. *How long is this wire?*), or first selects one or more entities and then speaks (*Which of these is longest?*).

While it does not appear possible to query QuickSet for the currently selected set of objects, each individual object selection and deselection results in a message being posted to the blackboard by the QuickSet BRIDGE agent, allowing NAUTILUS to "eavesdrop" on that message traffic and maintain a list of the POIDs of the currently selected object set. When NAUTILUS encounters a reference of the form *this/these (<noun>)* or *here* it queries the ModSAF database to retrieve the set of objects with those POIDs, which it then uses to dereference the expression.

SPEECH I/O

To accommodate the new language inputs to NAUTILUS, we implemented a new external speech recognition agent using our Windows/PC port of the Agent Library and the IBM ViaVoice commercial speech recognition package. The agent can either be triggered by a push-to-talk button in its graphical user interface, or the microphone can just be left open and the agent will send a `parse_speech` message to the blackboard whenever it thinks it has heard something covered by its recognition grammar. To aid in development work, we also implemented a text-input agent that allows sentences not currently covered by the speech recognizer to simply be typed in at the keyboard.



Finally, a speech output agent (also implemented in ViaVoice) accepts `speak_list` goals relayed to the blackboard by NAUTILUS and outputs them using text-to-speech synthesis. These utterances include not only responses to queries ("No, only M1A1 platoon number 3") but also error messages reporting unknown words ("I don't know word EXINIT"), parse failure ("Sorry, please rephrase") and type and

number errors (“No such platoon exists”). The speech agent graphical display includes an animated face whose mouth and features move while speaking and whose facial expression can be modified to express various emotional attitudes; we currently give it a “neutral” expression when answering queries, “surprised” when unable to understand the user’s input, and “puzzled” when reporting unknown words and number errors.

REFERENCE COVERAGE

Unlike the first (OGI_NL emulation) phase of the project, in the second phase (database query) we implemented full coverage of the QuickSet object domain. This section discusses some additional code macros that were developed to assist in this process, and some design philosophy issues that arose.

In NAUTILUS the mapping from a word sequence like *M1A1 company* to a ModSAF designator like `unit_USMC_M1A1_Company` takes place in three phases: (1) the word sequence is parsed and interpreted as a semantic type; (2) the semantic type is mapped to one or more typed objects; and (3) ModSAF designators are obtained from the typed objects’ feature lists. The reason for this multilevel approach is twofold: first, to correctly handle one-to-many and many-to-one correspondences between words and their semantics (for example an “M1A1 company” might be either a military echelon or an industrial firm that manufactures M1 tanks), and second, to separate out application-specific from domain-general information (in QuickSet it happens that the designator for the military echelon is “unit_USMC_M1A1_Company”).

As a result, extending the interface’s referential coverage is not simply a matter of providing NAUTILUS with an online lexicon, since such a dictionary (a) would be unlikely to contain military proper names and acronyms such as “M1A1”, (b) may or may not represent semantic synonymy or polysemy, e.g. that a “tank company” is something fundamentally different from an “arms company”, and (c) has no information about the final application-specific representation that is required. Hence NAUTILUS needs more than just a lexicon indicating that “M1A1” is a proper name, but also a semantics associating the name with a particular class of tank and a domain model describing the attributes of that object class, all of which must be largely hand-coded.

In practice, however, certain simplifying assumptions can be made that will allow partial automation of this model-building process. In particular, if we assume that (a) the phrase “M1A1 company” has only one parse and semantic interpretation and that (b) there is only one domain object type matching that semantics, then it becomes possible to automatically generate minimal semantic and domain-object structures from the phrase itself. We developed such a “supermodeling” facility in this project and used it experimentally to generate the NAUTILUS structures for one particular class of QuickSet entities, the cultural feature objects (airfields, barracks, church, etc.) For example the macro call

```
(supermodel (small vehicle maintenance building))
```

generates all the NAUTILUS data structures needed to map the phrase *small vehicle maintenance building* to a ModSAF point object of the same name. Note that under this approach we cannot parse and semantically analyze the noun phrase as “a small building for the maintenance of vehicles” (versus a building for the maintenance of small vehicles), but must treat it as a fixed idiom, allowing only for singular and plural variants. As a result, the FOCAL domain object cannot also be referred to as a “building” or a “vehicle maintenance building” since no such type hierarchy or modifier mappings have been established. It might be possible to automate the generation of a type hierarchy as well if we assume the modifier relationships to always be strictly right-branching, but this is not always the case: a “shallow draft vessel dock” is not also a “*draft vessel dock”, etc.

DISCUSSION

The NAUTILUS logical form was originally developed for interfacing to databases that do not possess their own feature-based query language, so it tends to represent everything as strictly predicative or functional rather than attributive. For example NAUTILUS would interpret the noun phrase “command posts with two jeeps” as the logical form

```
(SETOF X1 (SETOF N1 COMMAND-POST PLURAL)
       (EQ 2 (COUNT (HAVE X1 'JEEPS))))
```

which it would evaluate by making a database query to get the set `N1` of all command posts, evaluating the predicate `HAVE(<post>,jeeps)` for each returned command post object, and finally evaluating the `COUNT` and `EQ` functions on the resulting sets. In previous NAUTILUS applications each of the `HAVE` invocations would have required an additional database query to test whether the post in question had a `JEEPS` attribute value, but since the objects returned by QuickSet are typed feature structures that include the number of jeeps as an explicit feature, in this case NAUTILUS can perform the test itself. However, since the ModSAF Agent Layer language supports formula unification, “command posts with two jeeps” could alternatively have been retrieved by the single database query

```
list_points(point(_,_,_,command_post([jeeps(2),_,_]),_,_,_,_))
```

requiring no additional filtering or testing at the NAUTILUS end. NAUTILUS could generate such a query only if it interpreted the “with” phrase as attributive rather than predicative/functional, so this is a possible extension meriting further consideration.

By integrating deixis with graphical selection, we were able to maintain our usual practice of dereferencing deictic references during NAUTILUS processing (as with all other forms of reference), so that the logical forms for utterances like *How long are these wires?* and *How long are wires 1 and 2?* generate identical semantics. From the QuickSet user’s standpoint, however, a gesture-based treatment of deixis is probably preferable, particularly since graphical selection does not appear to play a role in the graphical interface itself. Under the QuickSet model of multimodal integration, the phrase *How long are these wires?* would instead generate a partial semantic structure requiring unification with the graphical gesture by a separate multimodal integrator agent (possibly the existing QuickSet integrator, though that is unlikely), with the resulting integrated feature structure then passed back to NAUTILUS for logical form conversion and evaluation.

NAUTILUS was also developed in environments where it was the only natural language processor present, so it typically issues spoken error messages for unknown words, parse failure and reference errors. In the QuickSet environment, however, the OGI_NL agent remains active alongside NRL_NL (to handle object creation and the other original QuickSet commands), so inputs not handled by one might be processable by the other, making a spoken error message from one misleading. In that case we either need to restrict NAUTILUS error messages to unspoken text (as is the case with OGI_NL error messages), or somehow coordinate between the two agents.

RELATED WORK

CommandTalk [Moore 1997], also developed for LeatherNet, is a spoken natural language interface (input only) to ModSAF and CommandVU. Like the present work it is implemented using the Open Agent Architecture and includes agents for speech recognition, NL, and contextual interpretation (the latter includes noun phrase and predicate resolution, which in our system is handled by the NRL_NL agent). The CommandTalk noun phrase resolver handles various individual object references such as “M1 platoon”,

“tank platoon” and “Charlie 4 5” as does NAUTILUS, but does not appear to handle quantification or relativization. As the name CommandTalk suggests, it can issue a wide variety of imperative commands, but apparently not queries.

REFERENCES

Philip R. Cohen, A.J. Cheyer, M. Wang, and S.C. Baeg (1994). an open agent architecture. In Working Notes, AAAI Spring Symposium Series, Software Agents, Stanford, California, pp. 1-8.

Philip R. Cohen, M. Johnston, D. McGee, S. Oviatt, J. Pittman, I. Smith, L. Chen and J. Clow (1997). QuickSet: Multimodal interaction for simulation set-up and control. Proceedings of the Fifth Conference on Applied Natural Language Processing, Mar. 31-Apr. 3, 1997, Washington DC, pp. 20-24.

R. C. Moore, J. Dowding, H. Bratt, J.M. Gawron, Y. Gorfu, and A. Cheyer (1997). CommandTalk: A spoken-language interface for battlefield simulations. Proceedings of the Fifth Conference on Applied Natural Language Processing, Mar. 31-Apr. 3, 1997, Washington DC, pp. 1-7.

Kenneth Wauchope (1994). Eucalyptus: Integrating natural language input with a graphical user interface. NRL Report NRL/FR/5510-94-9711, Naval Research Laboratory, Washington, DC, 39pp.

Kenneth Wauchope (1996). Multimodal interaction with a map-based simulation system. Navy Center for Applied Research in Artificial Intelligence Internal Report AIC-96-019, Naval Research Laboratory, Washington, DC, 21pp.

Kenneth Wauchope, S. Everett, D. Perzanowski, and E. Marsh (1997). Natural language in four spatial interfaces. Proceedings of the Fifth Conference on Applied Natural Language Processing, Mar. 31-Apr. 3, 1997, Washington DC, pp. 8-11.